Zeitschrift: Actes de la Société jurassienne d'émulation
Herausgeber: Société jurassienne d'émulation
Band: 106 (2003)

Artikel: Une brève histoire des logiciels libres
Autor: Fuhrer, Claude
DOI: https://doi.org/10.5169/seals-685025

Une brève histoire des logiciels libres

Claude Fuhrer

Les logiciels libres (*free software* en anglais) sont une alternative intéressante aux modèles standards de production des logiciels, c'est-à-dire les logiciels propriétaires. Leur but est de garantir aux utilisateurs une liberté qui leur a été depuis longtemps déniée par les grandes entreprises de logiciels.

La notion de logiciel libre?

Il serait peut-être utile de définir ce qu'est un logiciel libre. Malheureusement, il n'existe pas une seule définition du logiciel libre, mais plusieurs qui diffèrent chacune par leur interprétation de la liberté que l'on doit accorder à l'utilisateur de logiciel libre. Car, s'il y a bien un point sur lequel toutes ces définitions s'accordent, c'est que leur but est de garantir la liberté de l'utilisateur final.

Chacune de ces interprétations du terme de liberté a donné naissance à une licence logicielle, c'est-à-dire un document légal décrivant les droits et devoirs des utilisateurs de logiciel libre. Le terme est lui aussi contesté. Le terme français «logiciel libre» ne présente pas la même ambiguïté que le terme anglais «*free software*». Le terme de *free software* a été proposé la première fois en 1984 par Richard Stallman. Comme définition de ce terme il proposait la justification:

free as in free speech, not as in free beer

Les neufs commandements du logiciel libre

Certains théoriciens du logiciel libre ont posé neuf commandements qui définissent ce que devrait autoriser (et interdire) une licence vraiment libre.

Ces commandements sont :

1. Diffusion libre

La licence ne peut entraver la vente ou la propagation des logiciels libres. En particulier, il ne peut y avoir de paiement de *royalties* pour les produits dérivés d'un tel logiciel.

2. Code source

Un logiciel libre doit comporter le code source et permettre la distribution du code source aussi bien que la version compilée. Il n'est pas absolument nécessaire que le code source soit livré directement avec l'application, mais, dans ce cas il doit, par exemple, figurer sur un site Web ou il pourra être téléchargé gratuitement.

De plus, la licence doit autoriser l'utilisateur à employer ce code source, à le modifier et à en distribuer la forme modifiée.

3. Travaux dérivés du programme original

La licence doit autoriser la modification du programme original, mais doit aussi en même temps encourager (voire obliger ?) la distribution de ces modifications selon les mêmes termes que l'œuvre originale.

4. Préservation du code source original

Il faut autoriser la modification et l'amélioration du programme original, mais les différents auteurs d'un programme ont aussi droit à la protection de leurs talents et de leur réputation. La licence doit donc contraindre les développeurs à indiquer de manière claire les contributions de chacun et différencier (par exemple par un nouveau numéro de version) le programme modifié du programme original.

5. Aucune discrimination de personnes ou de groupes

Le logiciel libre doit être accessible à tout le monde. Il n'est en aucun cas permis de réduire sa disponibilité à un groupe de personnes comme par exemple n'autoriser qu'un usage éducatif mais pas industriel.

6. Aucune restriction relative au domaine d'utilisation

En dehors des clauses de non-responsabilité, le programmeur ne peut pas (ne doit pas) restreindre l'utilisation de son logiciel pour, par exemple, en interdire l'utilisation dans un domaine qui ne lui semble pas acceptable.

7. Transmission de la licence

Ceci est une des caractéristiques les plus importantes et les plus intéressantes du logiciel libre. Les droits attachés à un programme valent pour tous ceux qui ont obtenu ce programme, sans qu'il soit permis à un intermédiaire d'y attacher une licence supplémentaire ou de simplement modifier la licence originale.

8. La licence ne doit pas être attachée à un produit particulier

Il ne faut pas qu'un programme soit libre quand il est distribué d'une certaine manière et non libre dans un autre contexte. Cela permet de se prémunir des «pièges à licence».

9. La licence ne doit pas entraver d'autres logiciels

Cette clause est en quelque sorte le pendant de la précédente. On doit pouvoir distribuer des logiciels libres dans un environnement ou une distribution de logiciels propriétaires. En annexe au présent document, on trouvera une liste de références d'où il est possible de télécharger le texte complet de ces licences.

Quelques licences libres

Ainsi qu'il a été mentionné plus haut, il existe plusieurs licences «libres». Parmi celles-ci, trois ont pris le pas sur les autres. Elles couvrent la majorité des logiciels libres. Le tableau ci-dessous donne un petit aperçu de ces licences avec certaines de leurs particularités.

La licence BSD

La licence BSD (*Berkeley Software Distribution*) est une des plus anciennes mais aussi des moins contraignantes. Elle permet pratiquement toutes les utilisations du code qu'elle recouvre, y compris l'inclusion dans un programme propriétaire, ce qui fait d'ailleurs dire aux puristes qu'elle n'est pas une vraie licence libre. Cette licence impose de recopier dans les fichiers source un en-tête standardisé comportant les principaux termes de cette licence. Cette particularité conduit à des situations plutôt comiques dans laquelle un fichier source de quelques lignes doit inclure une en-tête de plusieurs centaines de lignes.

LICENCE	Utilisable avec un logiciel commercial	Accès aux modifications pour tous	Restrictions de publication	Droits particuliers pour le détenteur de la licence
GPL	non	oui	non	non
LGPL	oui	oui	non	non
BSD	oui	oui	non	non
NPL	oui	non	non	oui
Domaine public	oui	non	oui	non

La licence GNU-GPL

La licence GNU-GPL (*GNU General Public Licence*) est non seulement une des licences les plus répandues, mais aussi la plus intéressante. Outre les termes classiques de non-responsabilité, elle contient un certain nombre de clauses dont on pourrait dire qu'elles contraignent à la liberté.

La licence GNU-GPL, comme toutes les licences de logiciel libre, donne à l'utilisateur du logiciel qu'elle couvre la liberté d'utilisation, de copie et de modification. Par contre, où elle se distingue d'un certain nombre d'autres licences, c'est qu'elle oblige le détenteur d'un logiciel GPL à transférer ses droits aux autres utilisateurs. Cela signifie que si quelqu'un fournit à une autre personne un logiciel couvert par la licence GNU-GPL il doit obligatoirement donner le code source, y compris des modifications qu'il y a faites et cela dans les termes de la GPL. Cette clause a d'ailleurs souvent été qualifiée de «maladie virale» par une grande firme de développement de logiciel qui distribue uniquement du logiciel propriétaire.

Un peu d'histoire

Le mouvement du logiciel libre a particulièrement pris de l'ampleur au cours de ces dix dernières années. Cependant, ce mode de distribution n'est pas nouveau. Il était même souvent la règle au début de l'his-

toire de l'informatique. En effet, avant la «standardisation» des systèmes d'exploitation, les logiciels étaient beaucoup moins interchangeables d'un ordinateur à l'autre. Aussi, lorsqu'un programmeur avait développé un programme pour résoudre un problème, il était très heureux d'en faire profiter les autres utilisateurs du même type d'ordinateur. Souvent d'ailleurs, c'étaient les fabricants d'ordinateurs eux-mêmes qui rassemblaient et distribuaient ces utilitaires. Un exemple fameux est la fabuleuse collection de programmes mise à disposition des utilisateurs des ordinateurs de la gamme PDP par la société Digital Equipment Co. avec les célèbres «bandes DECUS».

Aussi, c'est tout naturellement que lorsqu'en 1969, deux programmeurs de la société AT&T décidèrent de concevoir et de réaliser un nouveau système d'exploitation, ils l'ont distribué avec le code source en espérant que les nouveaux utilisateurs puissent en profiter pour proposer des améliorations. De plus, il convient de remarquer que cette manière de faire permettait aussi à AT&T de ne pas enfreindre l'arrêt que la justice avait rendu quelques années auparavant en la condamnant pour infraction à la loi sur les monopoles. Ce nouveau système d'exploitation a été baptisé *Unix* et le langage de programmation qui a été développé conjointement est bien entendu le langage C.

A peu près à la même époque, dans le laboratoire d'intelligence artificielle du MIT, Richard M. Stallman développait des logiciels pour une machine de la firme Digital Equipment & Co. Cependant, l'ordinateur utilisé par le laboratoire arrivait en fin de vie et Digital avait cessé la production de systèmes compatibles avec cette antiquité. Le problème se posait donc de trouver une nouvelle machine, ce qui à l'époque signifiait aussi un nouveau système d'exploitation. Après avoir regardé à droite et à gauche, pour trouver ce qui se faisait dans les différents instituts de recherche, le laboratoire d'intelligence artificielle du MIT découvrit Unix.

Ce système d'exploitation a, par rapport à ses concurrents de l'époque, le gigantesque avantage d'être facilement portable d'un ordinateur à l'autre. Cela signifie que lorsqu'il faut changer de type d'ordinateur, les utilisateurs peuvent tout de même retrouver un environnement de travail familier. C'est donc dans ce contexte que Richard Stallman (rms pour les connaisseurs) commença à mettre en place ses idées de logiciel libre.

En 1984, AT&T prit enfin vraiment conscience du potentiel du logiciel Unix. Il n'était alors plus question de le distribuer gratuitement, mais bien de le vendre au prix fort, avec une licence plutôt restrictive. Bien entendu, rms était contrarié par ce changement d'orientation. Il décida d'écrire un clone de Unix, qui serait complètement libre. Ce clone comporterait non seulement le système d'exploitation mais aussi toute une collection d'outils et d'utilitaires qui permettraient de travailler plus confortablement et plus efficacement.

Quelques grands hackers

Avant de donner une courte biographie de quelques hackers célèbres, il faut mentionner que dans le monde du logiciel libre, on appelle hacker tout programmeur mettant sa propre production à disposition de la communauté. Un vrai hacker ne s'occupe pas de piratage de logiciels ou de sites informatiques. Ces activités illégales sont réservées aux *crackers*. Les vrais hackers n'ont d'ailleurs que peu d'estime pour les individus qui utilisent leurs talents pour violer la loi plutôt que de développer des outils qui pourraient être utiles à toute la communauté (voir la définition de ces deux termes dans le glossaire).

Don Knuth

Le professeur Donald Edwin Knuth a été un des premiers à adopter le principe du «logiciel libre» et cela avant même que le terme ne fût inventé. Au début des années 1960, le professeur Knuth, alors employé au département de mathématique de Caltech, décida qu'il était temps d'écrire l'encyclopédie définitive de l'art de la programmation des ordinateurs. Cette encyclopédie sera intitulée *The Art of Computer Programming* et est souvent référencée avec l'acronyme TAOCP. L'histoire même de cette publication est intéressante. Selon Knuth, elle devait comporter huit volumes. Il se mit donc au travail et fit paraître le premier volume en 1968. Le thème de ce premier volume est *Fundamentals Algorithms*. Après la parution de ce premier volume, il s'attela à la rédaction du deuxième volume intitulé *Seminumerical Algorithms*. Alors que le premier volume avait été imprimé avec des caractères en plomb, le second volume a été imprimé en utilisant un procédé électronique.

La typographie de cet ouvrage (pourtant paru en 1969, c'est-à-dire seulement une année après le premier) ne satisfaisait pas du tout Knuth. Il décida donc, avant de faire paraître le troisième volume, de développer son propre système de typographie numérique.

Pendant 3 ans, il a consacré son temps et son énergie à concevoir, programmer et mettre au point un traitement de texte qu'il a nommé TEX. Pour que ce logiciel puisse être utilisé par tous, Knuth en a fait don à l'*American Mathematical Society*. N'importe qui peut écrire/programmer un traitement de texte implantant les fonctionnalités de TEX. Pour obtenir la «certification», il suffit de faire passer au programme un certain nombre de tests conçus par Knuth lui-même et, s'il réagit correctement, alors on peut admettre que ce programme est un compilateur TEX officiel.

Les anecdotes concernant Don Knuth et TEX sont nombreuses. Cependant, une des plus significatives concerne la qualité du logiciel produit par Knuth. Lors de la première version, Knuth a proposé de donner une récompense à chaque personne qui, la première, trouverait une erreur dans un de ses livres décrivant le système TEX. La récompense serait doublée à chaque nouvelle erreur trouvée. Certes Knuth a bien signé quelques chèques, mais leur montant n'a jamais été très élevé et, de plus, bien peu ont été encaissés. Pouvoir afficher un chèque signé de la main de Knuth pour avoir trouvé une erreur dans son travail est pour un vrai hacker une récompense bien plus importante que les quelques dollars que le chèque représente.

Richard M. Stallman

Richard Stallman est vraiment le père fondateur du mouvement du logiciel libre.

L'histoire liant Richard Stallman au logiciel libre commence au début des années 1970 au laboratoire d'intelligence artificielle de l'institut de technologie du Massachusetts (MIT). A cette époque, le laboratoire utilisait un ordinateur PDP10 de la firme Digital Equipment Co. avec un système d'exploitation dénommé non sans une certaine malice *Incompatible Timesharing System (ITS)*. Chaque membre du laboratoire développait ses propres logiciels et partageait ses expériences avec quiconque le désirait. Richard Stallman aurait pu vivre alors une vie sans histoire si le laboratoire d'IA n'avait pas été démantelé quelques années plus tard et s'il n'avait pas été nécessaire de remplacer l'ordinateur vieillissant par un système plus moderne.

Mais qui dit système plus moderne dit aussi souvent nouveau système d'exploitation. Le système ITS ne fonctionnait plus sur le nouvel ordinateur sur lequel travaillait Stallman. Celui-ci se retrouva donc confronté aux systèmes d'exploitation propriétaires avec les contraintes qui les accompagnent souvent c'est-à-dire les accords de non-divulgation. Cette façon de voir l'informatique choqua profondément Stallman. Pourquoi ne pouvait-il pas faire profiter d'autres hackers de son travail?

Réciproquement pourquoi devait-il «réinventer la roue» à chaque fois, c'est-à-dire résoudre à nouveau des problèmes qui avaient déjà été résolus pas d'autres?

Pour pouvoir développer efficacement de nouveaux programmes il était, à l'époque du moins, nécessaire de pouvoir accéder au code source du système d'exploitation. Or, c'était précisément ce code source qui était couvert par les clauses de non-divulgation. Comment contourner le problème? La réponse s'imposa rapidement, il fallait que Stallman

écrive son propre système d'exploitation. Le plus moderne de l'époque étant le système Unix, il décida de rendre son système compatible avec Unix.

Afin de garantir sa liberté et la liberté de ce qu'il produisait, en janvier 1984, il quitta définitivement le MIT pour se consacrer à plein temps à la tâche qu'il s'était fixée. Fidèle à une tradition des hackers, il adopta comme nom pour son projet un acronyme récursif: *GNU* qui signifie *Gnu is Not Unix.*

Un nouveau système d'exploitation a besoin d'outils tels que des compilateurs, des assembleurs, des éditeurs de texte, des programmes de messagerie, des interpréteurs de ligne de commande et bien d'autres encore. Stallman s'est donc attelé à la tâche en estimant qu'il serait plus utile de commencer par programmer ces utilitaires que de s'attaquer au système d'exploitation proprement dit. Dans le même élan, il a commencé à définir ce que devait être un logiciel libre. Il devrait garantir au moins les quatre libertés suivantes:

1. liberté d'exécuter le programme;
2. liberté de modifier le programme pour pouvoir l'adapter à ses propres besoins. Pour cela il est nécessaire de pouvoir accéder au code source;
3. liberté de redistribuer des copies du programme (gratuites ou non);
4. liberté de distribuer des versions modifiées du programme afin de faire profiter la communauté des améliorations apportées.

Ces quatre libertés fondamentales sont les quatre piliers qui soutiennent la licence GNU-GPL.

La troisième liberté est souvent mal comprise, en partie à cause du double sens en anglais du mot *free*. Cependant il est clairement expliqué que rien n'interdit à un développeur de vendre un logiciel libre qu'il aurait développé. Par contre il ne peut pas non plus empêcher ses «clients» de redistribuer gratuitement le logiciel qu'ils auraient acheté.

En septembre 1984, il s'attelle à la programmation de l'éditeur de texte *Gnu-emacs*, dont il commence à vendre les premières copies au printemps 1985. Il faut se souvenir qu'à cette époque internet (du moins tel qu'on le connaît actuellement) n'existe pas et donc qu'il est impossible pour la plupart des utilisateurs d'aller simplement télécharger le programme comme on le ferait aujourd'hui. Il a ensuite développé le compilateur *gcc*, l'interpréteur de ligne de commande *bash* et toute une collection d'utilitaires divers.

Le copyleft, la licence GPL et la FSF

Pour s'assurer que son travail ne serait pas récupéré par d'autres qui pourraient en travestir l'esprit (comme cela s'est vu avec le logiciel

X-Windows, Stallman a mis au point une licence dont le but est de protéger les droits des utilisateurs.

Toujours dans l'esprit facétieux des hackers et en jouant avec le double sens de certains mots, cette nouvelle licence est censée définir une gauche d'auteur (*copyleft*). Pour donner une structure officielle au logiciel libre, une fondation a été créée; elle s'appelle FSF (*Free Software Foundation*).

Linus Torvalds

Un jour de juillet 1991, un jeune étudiant finlandais a posté dans le newsgroup comp. os. minix le message suivant ([TD01]):

Salut les netlandiens, suite à un projet sur lequel je travaille (sous Minix), je serais intéressé par la définition du standard Posix. Quelqu'un pourrait-il me dire où je pourrais trouver la version la plus récente des règles Posix dans un format de préférence lisible par une machine? Un site FTP serait très apprécié.

Ce message apparemment anodin est à la base d'une des aventures les plus originales et les plus fabuleuses de la courte histoire de l'informatique. L'auteur de ce message est Linus Torvalds, un étudiant en informatique de Finlande. Il semble que ce message ne suscita pas beaucoup de réponses. Linus dut donc trouver un autre chemin pour obtenir l'information dont il avait besoin. A peine un mois plus tard, le 25 août 1991, il a publié un autre message où il demandait aux utilisateurs de Minix quelles sont les fonctionnalités qu'ils désiraient voir ajouter à ce logiciel pour construire un nouveau système d'exploitation. Il s'écoula encore près de deux mois pour qu'il publie sa première version, numérotée 0.02. En informatique, on désigne l'évolution des logiciels par leur numéro de version. Le nombre à la gauche du premier point est appelé «numéro de version majeure» (*major version number*). Un logiciel dont le numéro de version majeur est 0 est un logiciel en cours de développement et qui n'est pas encore considéré comme vraiment utilisable. Le deuxième nombre est appelé «numéro de version mineure». Il est censé indiquer l'état des changements depuis le dernier numéro de version majeur. Quand on constate que le numéro de version mineur de cette version de Linux est 02, on peut imaginer la somme de travail que Linus Torvalds envisageait pour arriver à une version officielle. Il était encore cependant bien loin de la vérité. Le travail a duré plus de deux ans. Pendant cette période, Linus a reçu des milliers de rapports de *bogues* (ce qui n'a rien d'extraordinaire pour un développement de cette ampleur) et a produit un nombre incalculable de mises à jour. A cette époque, installer Linux sur un ordinateur relevait de l'exploit sportif. Il fallait tout

d'abord copier sur Internet le contenu de plusieurs dizaines de disquettes. Ensuite, l'installation se faisait manuellement en copiant au bon endroit les bons fichiers. Aucune aide non plus quant aux dizaines de paramètres qu'il fallait introduire dans les multiples fichiers de configurations sur lesquels repose Linux.

Jusque là, les ordinateurs fonctionnant sous Linux avaient pratiquement été conçus pour ce système d'exploitation. Ou alors, par exemple dans le cas du système Vax de Digital Equipment Co., le système Unix avait été développé spécialement pour l'ordinateur. Le matériel était connu, ainsi que la manière dont il réagissait. Pour la première fois, quelqu'un se proposait d'écrire un système d'exploitation pouvant fonctionner sur des machines aussi hétéroclites que sont les PC. On pourrait rétorquer que Microsoft à l'époque arrivait déjà à maîtriser cette prouesse technique. La différence fondamentale était quand même que Microsoft bénéficiait du support des fabricants de matériel informatique, ce qui n'était pas le cas (et de très loin) de Linux. Il a donc fallu comprendre comment fonctionnait chaque composant d'un ordinateur, chaque périphérique, écrire un logiciel de pilotage (*driver)* avec les informations obtenues. De plus, pour garantir la liberté de Linux, il n'était pas question d'utiliser un code fourni par un fabricant qui n'aurait pas été couvert par la licence GNU-GPL de Linux.

Actuellement, Linus continue à superviser le développement de Linux. Il est en particulier le principal superviseur des nouvelles versions du noyau. Cependant, il est amusant de constater que quand Linus Torvalds a décidé de chercher un emploi, il n'a pas choisi une activité directement en relation avec Linux, mais a décidé de travailler pour une petite firme presque inconnue qui développe des microprocesseurs basse consommation pour les systèmes portables.

Eric Raymond

Eric Raymond s'est autoproclammé le théoricien du logiciel libre. Sa «contribution logicielle» au monde du libre est surtout connue des spécialistes. Il a en particulier initié et supervisé le développement du logiciel *fetchmail* ainsi que de plusieurs modules d'extension du logiciel GNU Emacs de Stallman. Cependant, la raison qui a fait d'Eric Raymond un «personnage» dans le monde du logiciel libre est la publication d'abord sous la forme d'un article et ensuite sous la forme d'un livre ([Rey00]) d'une étude montrant les différentes techniques de développement que le logiciel a inventées, ou pour le moins appliquées, à une grande échelle, et mises en lumière.

Eric Raymond a appelé cette technique de développement la technique du bazar. Curieusement, l'apparent chaos qui préside à cette technique de développement amène rapidement une ligne de développement stable. De plus, comme la plupart des hackers (du moins tous ceux ayant des contributions significatives dans le développement de logiciels libres) s'imposent des standards de qualité que l'industrie peine à atteindre, les produits obtenus sont souvent meilleurs tant du point de vue de la conception que du point de vue de la fonctionnalité.

L'« Internet Engineering Task Force » (IETF)

L'IETF est un comité en charge d'établir les standards qui permettent aux différents ordinateurs connectés sur internet de s'échanger des informations de manière transparente et sûre pour les utilisateurs. Parmi les grandes créations de l'IETF on trouve entre autres le protocole TCP/IP qui régit les transferts d'information « bas niveau » sur internet, la « norme » RFC 822 qui définit le format des millions de mails qui transitent chaque jour entre les utilisateurs, et bien d'autres encore.

Le but de l'IETF est de définir des règles (ou plutôt des recommandations) permettant d'augmenter la sécurité et la qualité des échanges. Ce qui fait la force de l'IETF par rapport, par exemple, à d'autres comités de normalisation gouvermentaux, c'est que le processus d'élaboration des standards est complètement ouvert à toutes les personnes qui voudront bien s'y intéresser. En fait l'IETF n'a qu'une existence virtuelle car il ne dispose d'aucune liste de membres. Aucune cotisation, aucun critère géographique ou professionnel ne limite la participation de tout-un-chacun à ce comité. On considère normalement que toute personne abonnée à une des listes de diffusion de l'IETF, ayant participé ne serait-ce qu'une fois à une des réunions, ou ayant émis un commentaire sur une des recommandations, est automatiquement considérée comme membre officiel. D'ailleurs les travaux effectués le sont toujours de manière bénévole, sans autre salaire que le sentiment d'avoir « fait un bon boulot ».

Processus d'élaboration des standards

Il faut tout d'abord savoir que l'IETF est organisé en sections. Il y en actuellement huit qui sont :

1. applications ;
2. générale ;
3. internet ;

4. opérations et gestion;
5. routage;
6. sécurité;
7. transport;
8. service aux utilisateurs.

Chacune de ces sections est elle-même divisée en groupes de travail; il en existe plus d'une centaine.

La devise de l'IETF est *le consensus sur l'essentiel et du code qui fonctionne*, ce qui illustre bien le pragmatisme qui guide les décisions. Pour qu'une résolution soit prise, l'unanimité n'est pas nécessaire, mais on admet généralement que si moins de 80% des membres la soutiennent, il est plus prudent de la rejeter.

Si les règles de participation à l'IETF (cf. adresse du site Web à la fin de cet article) sont assez floues et souples, le processus d'élaboration de recommandations est lui très précisément codifié. Il se déroule en plusieurs phases dont les principales sont les suivantes:

1. Un groupe de travail commence par produire un brouillon internet (*internet draft*) proposant une solution à un problème intéressant les utilisateurs.

2. Le brouillon une fois élaboré est soumis une première fois à la communauté dans un processus appelé «dernier appel». Ce dernier appel s'étend sur une période de deux à quatre semaines.

3. La troisième phase peut être décomposée en deux parties distinctes. Tout d'abord, le document éventuellement corrigé acquiert pendant six mois au moins le statut d'*expérimental*. Au bout de cette période il peut éventuellement passer au statut de *pré-standard*. Pendant cette troisième étape, on dit que le standard est un *standard proposé*.

4. Pour pouvoir passer à la quatrième étape, il faut qu'au moins deux implémentations génétiquement indépendantes de la proposition aient été réalisées. De plus, ces deux versions doivent être pleinement interopérables. Dans la réalité, il est souvent nécessaire d'avoir un beaucoup plus grand nombre de programmes implantant le standard proposé pour faire la preuve de sa pertinence. Lorsque ces étapes ont été franchies, la proposition accède au statut de *standard internet*.

A partir de là, le nouveau standard va porter un numéro dans la prestigieuse série des RFC. Même si, à l'origine, les RFC étaient justement des propositions (*Request For Comment*), ce sont maintenant de vraies normes. Ici aussi, l'IETF se démarque des autres comités de normalisation en mettant gratuitement à disposition le texte de ces standards. Cette manière de procéder favorise leur diffusion et leur utilisation. En effet, dans d'autres organisations, l'accès aux standards exige de débourser d'importantes sommes d'argent, ce qui peut être un problème pour les petites entreprises ou les *startups* ainsi que les particuliers qui développent des outils en dehors de toute structure «industrielle».

Les méthodes du logiciel libre

Le logiciel libre a eu deux impacts importants sur les méthodes employées par les informaticiens pour développer et distribuer le logiciel. Ces deux points sont d'ailleurs intimement liés.

La base du logiciel libre consiste à fournir avec chaque logiciel libre le code source. Grossièrement, si l'on fait une analogie avec une œuvre musicale, on peut considérer que le code source est la partition alors que le code exécutable est le morceau de musique enregistré sur un CD. La plupart des utilisateurs se contentent de l'exécutable. Cependant, parfois, un utilisateur particulier désire une fonction spécifique, ou il lui semble nécessaire de corriger un *bug* qu'il trouve particulièrement gênant. S'il ne dispose pas du code source, il n'a pas d'autre solution que de demander au fournisseur du logiciel de bien vouloir modifier ou améliorer son produit, et attendre son bon vouloir. Alors que s'il dispose du code source et des connaissances nécessaires, il lui est possible d'effectuer la modification lui-même et de mettre ensuite celle-ci à disposition de la communauté. Une fois la modification effectuée et testée, deux choix s'offrent à lui. Le premier choix consiste à proposer à l'auteur «original» (ou plus exactement au chef de projet actuel) la modification pour qu'elle soit intégrée dans la version officielle. Il peut aussi décider de garder sa modification pour lui, et profiter ainsi du travail effectué par la communauté sans rien donner en retour.

Parfois, une troisième option se présente. Si le chef de projet rechigne à intégrer la modification proposée, soit parce qu'il lui semble qu'elle ne s'intègre pas dans l'orientation qu'il souhaite donner au projet, soit simplement parce qu'il n'a pas le temps de l'intégrer, alors le développeur peut proposer sa propre version du logiciel à la communauté. On aura alors deux produits «concurrents» quasiment identiques. Cet embranchement dans la vie d'un logiciel est appelé par le monde du libre un *fork*. Cette situation est le plus souvent malheureuse car cela signifie que les deux produits auront tendance à se «voler» des développeurs. Le cas s'est déjà présenté plusieurs fois. Les deux paires de produits les plus célèbres qui ont été issues d'un *fork* sont tout d'abord les éditeurs de texte *emacs* et *Xemacs* et ensuite les compilateurs pour les langages C et C ++ , *gcc* et *egcs*. Heureusement, dans ce deuxième cas, après quelques mois de vie parallèle, ils ont à nouveau fusionné en un seul produit.

Cet état de fait implique que n'importe qui peut alors modifier un programme et proposer de nouvelles fonctionnalités. Le concepteur original n'a alors plus qu'un contrôle restreint sur l'évolution de son produit. Jusqu'à l'apparition du logiciel libre, personne ne croyait qu'il soit possible de développer un programme sérieux selon ce modèle de partage total. La plupart des théoriciens de l'ingénierie informatique étaient

persuadés que la seule solution possible était le modèle qu'Eric Raymond a appelé le *modèle cathédrale*, c'est-à-dire que la vie du programme était sous la supervision et la responsabilité d'un «grand architecte» qui prenait seul les décisions d'extension du programme. Lui seul avait une vue d'ensemble suffisamment globale pour comprendre ce que le logiciel devait faire, pouvait faire, et surtout ne devait pas faire.

Le logiciel libre propose un nouveau modèle de développement des programmes. Eric S. Raymond propose de nommer ce modèle «modèle bazar». Comme tous les logiciels libres sont accompagnés de leur source, chacun peut décider quelles sont les améliorations possibles ou souhaitables. Tout utilisateur de logiciel libre peut aussi modifier les programmes pour y incorporer ces modifications. Ensuite, le modèle prévoit que l'on envoie une description des modifications au chef de projet qui va alors intégrer ces modifications dans la version «officielle». Cela signifie aussi que le produit va évoluer très vite. Et comme les multiples développeurs d'un projet sont le plus souvent très éloignés les uns des autres, il a fallu trouver des outils et des techniques qui permettent à chacun de communiquer avec les autres.

Il est évident que le monde du logiciel libre ne serait pas ce qu'il est aujourd'hui s'il n'y avait pas eu l'expansion d'internet.

On peut résumer les méthodes de développement en un certain nombre de règles dont les plus importantes sont:

1. *Chaque développement d'un projet sérieux commence par «gratter» une démangeaison du développeur.*

Il est inutile de commencer le développement d'un projet si on n'en voit pas réellement l'utilité, c'est-à-dire s'il ne résout pas un problème auquel est confronté le programmeur. D'ailleurs, la plupart des projets ayant échoué sont des projets qui ne répondaient pas à un réel besoin.

2. *Les bons programmeurs savent ce qu'il faut écrire. Les programmeurs géniaux savent ce qu'il faut réécrire et ce qu'il faut réutiliser.*

Il existe dans les bibliothèques de logiciels libres des millions de lignes de codes résolvant une très grande palette de problèmes. Il n'est pas toujours nécessaire de réinventer la roue. Parfois, il est beaucoup plus judicieux de réutiliser ce qui a déjà été écrit. Il arrive cependant que le développeur remarque une amélioration possible du code déjà écrit et dans ce cas il faut aussi savoir reconnaître que sa propre solution est réellement meilleure.

3. *Dans votre planification, prévoyez de jeter au moins une version complète de projet. Cela vous arrivera de toute façon.*

Aussi soigneusement que soit faite la spécification d'un programme et de son cahier des charges, il arrivera toujours un moment où il ne sera plus possible d'ajouter une fonction qui avait été oubliée et qui ne peut pas entrer dans la structure. La seule manière vraiment efficace de se rendre compte de ce genre d'oubli est de réaliser une première version

qui devra être considérée comme prototype et ensuite de recommencer le développement quand cela sera devenu nécessaire.

4. *Si vous avez la bonne attitude, les problèmes intéressants vont finir par vous trouver.*

5. *Quand vous perdez votre intérêt pour un programme, votre dernière obligation est de la transmettre à un successeur compétent.*

Dans quantité de projets de logiciel libre, le responsable du projet change au cours de la vie du projet. Cela est dû au fait que la plupart des logiciels libres sont développés bénévolement et que les personnes ayant initié le projet ont changé de profession, d'employeur ou tout simplement de priorité. Il existe sur le Web une liste de projets «à remettre», c'est-à-dire dont le responsable actuel cherche à passer la main. La plupart du temps, le nouveau responsable du projet est quelqu'un faisant un usage intensif du produit en question.

6. *Traiter vos utilisateurs comme des co-développeurs est le moyen le plus simple d'augmenter rapidement la qualité de votre code et de le débugger efficacement.*

Tout utilisateur du produit est susceptible de formuler des suggestions intéressantes et intelligentes et cela même si l'utilisateur en question n'est pas un programmeur.

7. *Release early. Release often.* Et soyez à l'écoute de vos clients.

8. *Etant donné un ensemble suffisamment grand de bêta-testeurs et de co-développeurs, tous les problèmes seront localisés rapidement et les corrections seront évidentes pour un des développeurs.*

9. *Une structure de données intelligente et un code stupide fonctionnent toujours mieux que l'inverse.*

10. *Si vous considérez vos bêta-testeurs comme votre ressource la plus précieuse, en retour ils feront ce qui est nécessaire pour devenir votre ressource la plus précieuse.*

11. *La deuxième chose la plus importante à avoir après de bonnes idées est de reconnaître les bonnes idées de vos utilisateurs.*

12. *Souvent la solution la plus originale et innovante à un problème est de voir que la compréhension originale du problème n'était pas bonne.*

13. *La perfection dans la conception n'est pas atteinte quand il n'y a plus rien à ajouter mais quand il n'y a plus rien à retirer.*

Les enjeux du futur

Prétendre que l'informatique, et plus particulièrement l'informatique personnelle, va prendre une part toujours plus importante dans notre vie quotidienne est une évidence. Le commerce électronique pour les loisirs

et les biens de divertissement (tels que les livres, les CD et les DVD) est en train d'exploser. De plus, avec l'arrivée des technologies telles que ADSL ou de l'accès internet par le câble du téléréseau, les ordinateurs des particuliers sont maintenant connectés en permanence à la toile mondiale. Il est donc indispensable de protéger les données contenues dans ces PC, ainsi que les transactions qui sont faites à partir de ceux-ci, en utilisant des logiciels de cryptographie ou de filtrage des accès.

La pratique montre que, dans le domaine de la cryptographie civile en tout cas, la meilleure fiabilité est atteinte par des programmes qui non seulement implantent des algorithmes publics, mais dont le code source est aussi disponible. Cette disponibilité permet à tout utilisateur qui le désire de s'assurer que le programme ne contient pas d'erreurs qui compromettent la sécurité que le logiciel est censé fournir. C'est typiquement un domaine dans lequel le logiciel libre a montré sa supériorité par rapport au logiciel propriétaire. Bien entendu, il faut aussi résoudre le problème de la distribution du logiciel. C'est-à-dire qu'un utilisateur qui n'aurait pas le temps ou les connaissances nécessaires à l'examen du code du programme doit pouvoir se procurer le logiciel depuis une source qu'il considère comme fiable.

Une autre application de l'informatique est en train d'émerger et qui pourrait avoir des effets retentissants sur notre vie de tous les jours, c'est le vote électronique. Plusieurs cantons sont en train d'examiner la possibilité de remplacer les bulletins papiers par des systèmes informatiques dans le but d'accélérer et de simplifier les procédures de dépouillement. Ici aussi, l'utilisation d'un logiciel propriétaire pourrait avoir une influence catastrophique sur les principes de base de la démocratie. Imaginons seulement ce qui pourrait se passer si, après chaque week-end de votation, une entreprise ramassait, dans chaque commune, les urnes, pour délivrer quelques heures plus tard un résumé des votes. Cela semblerait totalement anti-démocratique. Pourtant c'est exactement la situation qui risque de se produire si les communes ou les cantons portent leur choix sur un logiciel propriétaire. Il ne sera alors plus possible aux citoyens de s'assurer que les procédures de dépouillement se dérouleront de manière juste et conforme à la loi.

Glossaire

Bêta-testeur : utilisateur d'un programme en phase *bêta*. La vie d'un programme passe tout d'abord par une phase appelée *alpha* pendant le développement.

Dans cet état, seules certaines fonctions du programme sont utilisables et uniquement par les développeurs du programme. Ensuite,

lorsque toutes les fonctionnalités planifiées ont été réalisées, on dit que le programme passe en phase *bêta*. Pendant cette période, le programme est testé par des utilisateurs choisis pour leur compétence. Leur but est de débusquer les erreurs qui subsistent dans le programme et qui n'auraient pas été décelées par les programmeurs. Enfin, lorsqu'on estime que le programme est suffisamment débuggé, il passe alors en phase de production, appelée aussi version stable, c'est-à-dire qu'il est disponible pour tous les utilisateurs.

Binaire : les fichiers binaires sont des fichiers exécutables, issus de la phase de compilation d'un programme. Ils permettent, lorsqu'ils sont exécutés, de démarrer une application, comme un traitement de texte, un logiciel de messagerie, etc.

Compilateur/*Compiler* : pour réaliser une application, on utilise un éditeur de texte pour la saisie du code source, puis un compilateur pour le transformer en instructions binaires compréhensibles par la machine.

***Cracker* :** programmeur utilisant son talent de casser les protections mises en place par les éditeurs de logiciels propriétaires pour empêcher ou du moins limiter la copie de leurs produits. Un des principes de base des logiciels libres étant d'autoriser les utilisateurs à copier sans restriction, l'activité de *cracker* perd toute sa raison d'être. Les *crackers* se décrivent souvent comme des *hackers*, mais les vrais *hackers* ne les considèrent pas comme faisant partie des leurs.

Démon/*daemon* : programme Unix tournant en tâche de fond, c'est-à-dire sans être relié directement à un terminal.

Distribution : linux n'est qu'un noyau. Pour en faire un système complet, il est nécessaire de collecter un ensemble de programmes à travers Internet, et de les organiser pour obtenir un système fonctionnel. Des distributions (Red Hat, Mandrake,...) contiennent le noyau, ainsi que tous les programmes nécessaires au fonctionnement de l'environnement GNU/Linux.

Droits d'accès dans un système d'exploitation multi-utilisateurs, il est impératif que les fichiers des utilisateurs soient protégés les uns par rapport aux autres. Les droits d'accès déterminent les utilisateurs autorisés à lire, à modifier et à exécuter des fichiers.

Editeur de texte : il s'agit d'un programme qui permet d'éditer des fichiers au format texte, sans formatage ni gestion des polices de caractères. Il permet, par exemple, de modifier des fichiers de configuration ou des programmes sources.

***Freeware* :** le nom *freeware* joue avec le double sens que la langue anglaise attribue au mot *free*. Cependant, le mot *freeware* désigne la plupart du temps un logiciel gratuit, mais qui n'est pas forcément libre. Dans ce cas on trouve la dénomination française « graticiel ».

GNU : acronyme récursif signifiant *GNU is Not Unix*. Le mouvement GNU, issu de la Free Software Foundation a pour but de développer des

logiciels libres pour le plus grand nombre de systèmes informatiques possibles.

GNU/Linux: voir Linux.

***Hacker*:** la définition du *hacker* est assez floue. Suivant le dictionnaire ou la source, on trouve chaque fois une définition apparemment différente, mais qui souvent décrit la même chose. Voici quelques définitions:

1. une personne qui aime explorer les détails des systèmes programmables et comment étendre leurs capacités, par opposition à la plupart des utilisateurs qui préfèrent apprendre seulement le minimum nécessaire.
2. une personne qui programme avec enthousiasme (même obsessivement).
3. une personne qui est capable d'apprécier les *hack values*.
4. une personne qui est bonne en programmation rapide.
5. un expert dans un programme particulier, ou quelqu'un qui travaille fréquemment avec; par exemple un *Unix hacker*.

***How-to*:** il s'agit d'une collection de documents ayant trait à de nombreux sujets très particuliers. Ceux-ci sont généralement rédigés par des bénévoles à travers le monde et font office de mode d'emploi sur l'ensemble des problématiques concernant Linux.

IETF: *Internet Engineering Task Force*, comité bénévole en charge de normaliser internet, c'est-à-dire d'établir des standards qui permettent l'interopérabilité des différentes plate-formes connectées sur internet (voir RFC).

Journal/*Logfile*: les fichiers journaux sont des comptes rendus de l'activité d'une application, générés automatiquement et à intervalles réguliers. Ils peuvent être consultés à tout moment, notamment pour déceler l'origine d'un problème survenu lors de l'exécution d'un programme.

Linux: Linux est le système Unix développé à l'origine par Linus Torvald. Il a été le premier projet informatique d'envergure développé selon le principe du bazar.

Matériel/*Hardware*: tout ce qui concerne la partie «électronique» de l'informatique.

Noyau/*Kernel*: le noyau représente le système d'exploitation. Tous les accès au matériel, que ce soient la mémoire ou les périphériques, se font via le noyau. Ce dernier a également la charge de l'ordonnancement des processus, c'est-à-dire la gestion des différentes commandes effectuées.

Paquetage/*Package*: ce sont des fichiers d'archives permettant la distribution de logiciels, sous forme de fichiers source et/ou de binaires (ou exécutables). Ils se chargent de la gestion de dépendances logicielles par le biais de bases de données locales sur les packages préalablement installés (RPM, Debian, etc.)

POSIX: ensemble de normes définissant le comportement d'un système Unix.

Processus/*Process*: tout programme exécuté est un processus.

RFC: norme définissant les principaux standards d'internet. Bien que leur nom provienne de l'abréviation *Request For Comment*, ce sont bien des normes achevées et implantées par des dizaines de logiciels.

***Root*:** aussi connu sous le nom de «super utilisateur», ou administrateur du système, c'est le compte qui a tous les droits sur la machine à l'opposé des autres utilisateurs dont l'accès est limité.

RPM: système de paquetages très puissant, développé par la société Red Hat. C'est actuellement le système de *package* le plus répandu, mais il existe une alternative chez Debian.

Serveur mail: logiciel permettant de recevoir le courrier électronique en provenance d'autres ordinateurs et de le mémoriser en attendant que le destinataire le consulte à l'aide d'un client mail.

Logiciel/*Software*: tout ce concerne les programmes.

***Uptime*:** durée pendant laquelle une machine ou un logiciel fonctionne sans devoir redémarrer.

Quelques liens intéressants

- **http://www.linux-gull.ch:** le portail du Groupe des Utilisateurs de Linux et du Logiciel Libre Le Long du Léman (GULLLLLL).
- **http://www.gutenberg.eu.org Gutenberg:** le groupe des utilisateurs francophones de TEX.
- **http://www.info.fundp.ac.be/ofburlet/travaux/travail/travail.**: *html Hackers* et *crackers*: une même face cachée de l'informatique?
- **http://www.opensource.org/licences/gpl-licence.html:** la licence GNU-GPL.
- **http://www.ietf.org/:** le site officiel de l'IETF, l'organisme chargé des standards d'internet.
- **http://www.ietf.org/rfc.html:** le site contenant les différentes normes (RFC) définies par l'IETF et qui fixent le fonctionnement d'internet.
- **http://www.linux-france.org/article/these/cathedrale-bazar/:** la traduction française de l'article d'Eric Raymond qui à été la base du livre [Rey00].
- **http://www.oreilly.fr/divers/tribune-libre/index.html:** le site officiel du livre *Tribune libre*. Il est possible de lire la quasi-totalité de ce livre en ligne.
- **http://www.linux-france.org/prj/jargonf/index.html:** le jargon français: quelques définitions en français de termes utilisés par les *hackers* francophones.

• **http://www.ccil.org/jargon/:** le jargon file d'Eric Raymond. Ce site est le lexique de référence (en anglais) du monde des *hackers*.
• **http://www.stallman.org/rms.html:** la page *web* de Richard M. Stallman.
• **http://www.gnu.org/philosophy/categories.html:** la définition des différentes familles de logiciels (selon les droits qui y sont attachés) de la Free Software Foundation.

Références

[DOS99] Chris DiBona, Sam Ockman, and Mark Stone, editors. *Tribune libre : ténors de l'informatique libre*. O'Reilly, juin 1999.

[Inc01] O'Reilly Associates Inc. *Le logiciel libre précis et concis*. O'Reilly, 2001.

[Rey00] Eric S. Reymond. *The Cathedral and the Bazaar*. O'Reilly, 2000.

[TD01] Linus Torvalds and David Diamond. *Il était une fois Linux : l'histoire extraordinaire d'une révolution accidentelle*. Osman Eyrolles Multimedia, juin 2001.

Claude Fuhrer (Courgenay) est professeur d'informatique à la Haute Ecole Bernoise pour la Technique et l'Informatique (HTI) à Bienne.